# Classificação de alimentos segmentados

Lucas Santana Lellis - R.A.: 69618
Instituto de Ciência e Tecnologia
Universidade Federal de São Paulo

## I. Introdução

Oproblema das 8 rainhas consiste em posicionar 8 rainhas em um tabuleiro 8x8 de forma que elas não se cruzem em nenhuma direção.

Trata-se de um problema bastante complexo para uma busca em profundidade, já que são possíveis 4.426.165.368 estados diferentes, porém apenas 92 soluções. [1]

Existem diversas formas de se resolver este problema, incluindo soluções de inteligência artificial, como a **subida de encosta** e os **algoritmos genéticos**.

### A. Subida de encosta

Segundo [2], os algoritmos de subida de encosta baseiam-se em um loop simples a fim de maximizar( ou minimizar ) um valor. Trata-se de uma busca gulosa, que tem como condição de parada a descoberta de um valor máximo( ou mínimo ) local ( ou global ), eles param de iterar.

O valor, representado pela "colina", pode ser a **função de utilidade** que define a solução do problema em questão. A cada passo são avaliados os diferentes estados possíveis, sendo escolhida sempre a de maior utilidade.

Este algoritmo apresenta diferentes problemas, como:

- O **máximo local**, (ou mínimo local), em que o valor encontrado não é a solução ideal do problema.
- As **cordilheiras** são sequencias de máximos locais, o que torna difícil a navegação para algoritmos gulosos.
- Os **platôs** são áreas planas da topologia de espaço de estados. Nestes trechos o algoritmo fica parado, uma vez que não existe um caminho superior para que o algoritmo continue a subir.

Por estes motivos, este algoritmo possui uma alta taxa de erros, embora seja eficiente quando acerta.

No caso do problema das 8 rainhas, pretende-se **minimizar** o número total de colisões entre as rainhas.

### B. Algoritmo Genético

Segundo [2], um algoritmo genético é uma variante do algoritmo em feixe estocástica na qual os sucessores são gerados a partir da combinação de dois estados pais.

Este algoritmo baseia-se em conceitos de seleção natural, em que os seres evoluem a partir do cruzamento e mutação dos genes. Estes problemas baseiam-se em conceitos como "variabilidade genética", e de que "os melhores sobrevivem".

O algoritmo possui três etapas principais:

- **Geração de problemas:** É o passo em que a população inicial é gerada randomicamente.
- **Cruzamento:** O cruzamento entre cromossomos, em que os vetores que os representam são misturados em um $k$.
- **Mutação:** A mutação altera a informação em alguma parte do vetor, promovendo variações que podem ajudar a alcançar resultados inacessíveis a partir dos estados atuais.

Este algoritmo também procura maximizar uma função de utilidade, e pode ser implementado com diferentes abordagens. São José dos Campos, 24 de Abril de 2015

## II. Objetivos

Este projeto tem como objetivo implementar e comparar os algoritmos de **subida de encosta** e **genético**. Comparando também diferentes configurações do algoritmo genético.

## III. Metodologia Experimental

O projeto foi implementado inteiramente em *Python*, de forma que ambos os algoritmos compartilhassem dos mesmos geradores de problemas (**generate**), e das mesmas funções de utilidade (**collisions**). Tais funções estão presentes no arquivo "*lucaslellis_69618_common.py*".

### A. Algoritmo de subida de encosta

O algoritmo de subida de encosta foi feito segundo a especificação de Russel [2], e se dá, basicamente de acordo com o algoritmo 1. O problema foi divido em diferentes funções para fragmentar o problema, sendo estas:

- **Sucessors:** Função que recebe estado atual e retorna um vetor com todos os estados possíveis a partir do estado atual. Os estados são gerados somando ou subtraindo a posição de cada uma das rainhas.
- **Best:** Função que recebe o vetor de estados sucessores. Ela calcula e compara o número de colisões de cada um dos estados, e retorna o estado com o menor número de colisões.

O código referente ao algoritmo está presente no arquivo "*lucaslellis_69618_hillclimb.py*".

### B. Algoritmo genético

O algoritmo genético é um problema complexo e foi modelado utilizando classes. Após diversas tentativas em torná-lo robusto, a implementação final diferencia-se da definida por Russel [2] em alguns aspectos.

Primeiramente, a etapa de cruzamento gera dois descendentes, ao invés de um. Além disso, o algoritmo mantém uma porcentagem dos melhores genes da geração anterior, garantindo a evolução dos mais fortes, e reduzindo a chance de uma piora

**Algorithm 1** Algoritmo de Subida de Encosta

```
1: n = 8
2: vec = generate(n)
3: while collisions(vec) ≠ 0 do
4:     s = sucessors(vec)
5:     b = best(s)
6:     if ( collisions(b) ≤ collisions(vec) ) then
7:         break
8:     else
9:         vec = b
10:    end if
11: end while
```

nos resultados. Por último, os cruzamentos não são limitados por uma taxa modificável, uma vez que são realizados para complementar o espaço deixado pelos genes "mortos"da geração anterior.

O algoritmo se dá, basicamente de acordo com o algoritmo 2. O problema, ainda mais complexo, foi fragmentado em diversas funções, listadas a seguir:

- **crossOver:** Função responsável pela mistura de dois genes (vetores), dado um ponto k.
- **mutate:** Função utiliza um número aleatório entre 0 e 7 para definir uma posição do vetor, e substitui o valor nesta posição por um outro valor aleatório diferente do atual.
- **calcFitness:** Calcula o valor de 28 - crossover, uma vez que para o algoritmo genético procuramos maximixar uma função de utilidade.
- **fillPopulation:** Função responsável por gerar um vetor de genes arbitrários.
- **sortPopulation:** Função responsável por ordenar um vetor de genes em ordem decrescente de utilidade.
- **updateFitness:** Atualiza a utilidade de toda a população.
- **select:** Seleciona dois genes diferentes de uma população.

Além disso, existem algumas variáveis configuráveis:

- **populationSize:** Tamanho fixo da população.
- **keepRate:** Número de "sobreviventes"à cada geração.
- **mutationRate:** Probabilidade da mutação.
- **maxIterations:** Número máximo de gerações.

O código referente ao algoritmo está presente no arquivo "*lucaslellis_69618_genetic.py*".

### *C. Análise estatística*

Após implementados e testados, os algoritmos foram executados exaustivamente para a obtenção de dados para posterior comparação. Foram anotados o número de passos ( ou gerações ) necessários para finalizar o processamento, e o número de colisões encontrados ao final. O número de colisões ser maior do que zero significa que a simulação falhou em chegar a um resultado.

Para encontrar a melhor configuração do algoritmo genético, o programa foi executado 500 vezes com 180 diferentes combinações de parâmetros, totalizando 90000 execuções.

**Algorithm 2** Algoritmo Genético

```
Require: populationSize, keepRate, mutationRate, maxIterations
1: population = []
2: fillPopulation(population,populationSize)
3: updateFitness(population)
4: sortPopulation( population )
5: newPopulationSz = int(populationSize * keepRate) + int(populationSize * keepRate)%2 //Número par
6: finished = false
7: boardSize = 8
8: crossPoint = rand(0,boardSize)
9: count = 0
10: maxFitness = 28
11: while finished == false do
12:     count = count + 1
13:     crossPoint = (crossPoint+1)%boardSize population = population[0 : newPopulationSz ]
14:     while length(population) < populationSize do
15:         gene1, gene2 = select(population)
16:         gene1, gene2 = crossOver(gene1,gene2,crossPoint)
17:         population.append(gene1)
18:         population.append(gene2)
19:     end while
20:     for gene in population do
21:         if random(0,100) < mutationRate then
22:             mutate(gene)
23:         end if
24:     end for
25:     updateFitness(population)
26:     if fitness(best) == maxFitness or count ≥ maxIterations then
27:         finished = true
28:     end if
29: end while
```

Para limitar o tempo de execução, o algoritmo foi limitado a 1000 gerações.

## IV. Resultados e Discussão

Primeiramente, o algoritmo genético foi testado exaustivamente até que fosse encontrado o comportamento mais robusto e eficiente. O experimento foi feito fazendo 500 execuções para 180 diferentes combinações de variáveis.

Foram feitos testes para populações de 10, 30, 50, 80, 100, 250, 500, 750, e 1000 cromossomos, taxas de sobrevivência (keepRate) 25%, 50%, 75%, e 100% e taxas de cruzamento (crossRate) de 0% 25%, 50%, 75%, e 100%. E os 10 melhores resultados, ordenados pelo número de gerações, podem ser vistos na tabela I. Os dados completos do experimento estão disponíveis no arquivo "*genetics.csv*".

Analisando os dados, foi definido que a melhor combinação de parâmetros é a do experimento 144, em que o problema era resolvido em $7.71 \pm 4.08$ gerações. O baixo desvio pa-

Tabela I: Melhores resultados do algoritmo genético.

| Experiment | Pop. Size | Keep Ratio | Mutation Ratio | Generations |
|---|---|---|---|---|
| 144 | 750 | 0.25 | 0.75 | 7.71 |
| 165 | 1,000 | 0.25 | 1 | 7.91 |
| 164 | 1,000 | 0.25 | 0.75 | 8.69 |
| 145 | 750 | 0.25 | 1 | 9.55 |
| 163 | 1,000 | 0.25 | 0.5 | 9.74 |
| 167 | 1,000 | 0.5 | 0.25 | 10.73 |
| 124 | 500 | 0.25 | 0.75 | 10.92 |
| 125 | 500 | 0.25 | 1 | 11.59 |
| 147 | 750 | 0.5 | 0.25 | 12.28 |
| 168 | 1,000 | 0.5 | 0.5 | 12.29 |

drão mostra a estabilidade do algoritmo dados os parâmetros escolhidos.

Definido o melhor algoritmo genético, ambos os algoritmos foram executados 500 vezes, sendo obtida uma média de iterações/gerações, e uma taxa de erros. Tais resultados podem ser vistos na tabela II.

Tabela II: Comparação dos algoritmos.

| Algorithm | Steps | Collisions | Error Ratio |
|---|---|---|---|
| H illclimb | 4.06 | 1.25 | 84.6 |
| Gnetic | 7.96 | 0 | 0 |

Neste experimento, o algoritmo de subida de encosta precisa de $4.14 \pm 1.01$ passos para acertar, e $4.05 \pm 0.92$ passos para errar. Analisando tais dados, vemos que o algoritmo de subida de encosta, embora eficiente, possui uma altíssima taxa de erros, enquanto o algoritmo genético é estável, e dificilmente erra, pois não fica preso a um máximo local, e possui uma variabilidade bem maior, visto que existem mutações e grandes populações a cada passo.

## V. Conclusão

Os algoritmos de subida de encosta, embora eficientes para acertar um resultado, possuem uma altíssima taxa de erros, não sendo útil em situações em que o máximo global não é conhecido. Em situações em que o máximo local é conhecido, é possível reiniciar o algoritmo até que se encontre o resultado esperado.

Enquanto isso, os algoritmos genéticos possuem uma baixíssima taxa de erros, embora sejam complexos e difíceis de configurar. Sua complexidade também gera uma grande utilização de recursos computacionais, podendo levar horas para encontrar os parâmetros ideais para resolver um problema.

Portanto, a baixa taxa de erros permite que o algoritmo genético dê resultados mais fiéis ao esperado, e é melhor indicado para a solução do problema das 8 rainhas.

## Referências


[1] Wikipedia, Eight queens puzzle - Disponível em: http://en.wikipedia.org/wiki/Eight_queens_puzzle

[2] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25, 1995.